Java Programming 2 – Lab Sheet 2

This Lab Sheet contains material based on Lectures 3—4.

The deadline for Moodle submission of this lab exercise is 4:30pm on Thursday 10 October 2019.

Aims and objectives

- Writing more complex Java methods
- Using arrays and advanced control structures

Submission material

Please refer to lab sheet #1 for instructions on launching and using JShell.

Task 1: Computing Scrabble scores

Your task is to implement a method **`computeScore`** that correctly computes and returns the score that you would receive in the game of Scrabble[1] for the string given as a parameter – that is, you must add up the score of each letter in the string and return the total value. Refer to the following table of values for each letter:

A: 1	B: 3	C: 3	D: 2	E: 1
F: 4	G: 2	H: 4	I: 1	J: 8
K: 5	L: 1	M: 3	N: 1	O: 1
P: 3	Q: 10	R: 1	S: 1	T: 1
U: 1	V: 4	W: 4	X: 8	Y: 4
Z: 10				

If you encounter any characters other than those 26 letters, then they should not affect the total score, but **you should be sure to consider both upper case and lower case letters**.

The signature for the method must be as follows:

```
int computeScore (String word)
```

Sample outputs

Some possible results of running the method are as follows; note that we will test more cases when running your code.

```
jshell> computeScore ("")
$5 ==> 0

jshell> computeScore ("123&$%£&^%*^&")
$6 ==> 0

jshell> computeScore ("foo")
$7 ==> 6
```

[1] https://en.wikipedia.org/wiki/Scrabble or http://www.scrabble.com/

```
jshell> computeScore ("foo, bar")
$8 ==> 11

jshell> computeScore ("foO BAR")
$9 ==> 11
```

Your code will be marked automatically, so be sure that your method signature and output are exactly as shown above.

Hints and tips

Hint 1: The following methods of the **String** class may be useful for your implementation.[2] Assume that **`str`** is a variable of type **String**. Then …

- **`str.toCharArray()`** returns a **`char[]`** corresponding to the characters in **`str`**
- **`str.length()`** returns the length of **`str`** in characters
- **`str.charAt(i)`** returns the character at position **`i`** in **`str`**
- **`str.toLowerCase()`** returns a new String corresponding to **`str`** but with all letters in lower case, and **`str.toUpperCase()`** does the same thing but with upper case.

Hint 2: use a **`switch`** statement to determine the value for each letter in the string.

What to submit

Your final **computeScore** method should be stored in a file called **computeScore.java**

[2] Full documentation of the **String** class can be found at https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/String.html

Task 2: Validating credit card numbers

Your task is to write a program to check whether a given credit card number is valid or invalid. You can read more about the process of checking credit card numbers at several websites, including http://www.validcreditcardnumber.com/; the description below is a summary of the properties that we will be using for this lab.

The first several digits of the card number indicate the issuing network of the card:

- **Visa** card numbers all start with **4**
- **American Express** card numbers all start with **34** or **37**
- **MasterCard** card numbers all start with a number between **51 and 55 (inclusive**), or with a number between **2221 and 2720 (inclusive)**

The remainder of the card number is allocated by the card issuer. In general, a valid credit card number can be anywhere between 13 and 19 digits; however, depending on the issuer, the valid lengths can vary:

- All **Visa** card numbers are of length **13, 16 or 19**
- All **American Express** card numbers are of length **15**
- All **MasterCard** card numbers are of length **16**

Finally, the final digit of all credit card numbers is a **check digit** – that is, a digit computed from the other digits of the card number. Including this digit means that simple errors in manually entering a card number (such as a single mistyped digit or permutations of successive digits) will be caught because the card will not be validated. For credit cards, the check digit is computed using the Luhn Algorithm (https://en.wikipedia.org/wiki/Luhn_algorithm).

To test whether a credit card number is valid, proceed as follows (adapted from the Wikipedia page above):

1. **Proceeding from the rightmost digit and moving left**, double the value of **every second digit**. If the result of this doubling is greater than 9, then subtract 9 from the product.
2. Take the sum of the resulting digits.
3. The credit card number is valid if and only if the sum is a multiple of 10.

As a concrete example, consider the number “79927398713”. The validation process would proceed as shown in the following table.

Account number	7	9	9	2	7	3	9	8	7	1	3
Double every other	7	18	9	4	7	6	9	16	7	2	3
Digits to sum	7	9	9	4	7	6	9	7	7	2	3

The sum of the resulting digits is 7+9+9+4+7+6+9+7+7+2+3=70, so the number is valid according to the Luhn algorithm.

What you need to do

You must implement a method to validate a credit card number. The method signature should be as follows:

```
boolean checkCardNumber (String cardNumber)
```

Your method should carry out all of the checks listed on the previous page and should return **true** if the card number is valid and **false** if it is not. **Your code will only be tested on numeric strings, so you do not need to do any checking on the input except as specified above.**

You can use the sample (fictional) credit card numbers from websites such as http://www.getcreditcardnumbers.com/ to test your code. Be sure to test on both valid and invalid numbers!

Sample outputs

Here are some sample runs from the method. Note that we will use more tests when marking your code.

```
jshell> checkCardNumber ("371449635398431")
$25 => true

jshell> checkCardNumber ("371449635398432")
$26 => false

Jshell> checkCardNumber ("5444009197548666")
$27 => true

jshell> checkCardNumber ("123")
$28 => false

jshell> checkCardNumber ("51123")
$29 => false

jshell> checkCardNumber("2223000048400011")
$30 => true

jshell> checkCardNumber("4917610000000000003")
$31 ==> true
```

Hints and tips

Hint 1: The same **String** methods mentioned above for Task 1 can also be useful for your implementation here. You might also want to use the following (again, assume **str** is a String):

- **`str.substring(i, j)`** returns a new **String** which is the set of characters beginning at index **`i`** and ending at index **`j-1`**

Hint 2: You will probably also find the following methods useful for converting to integers:

- **`Character.getNumericValue(c)`** converts a **char** value **c** into its corresponding integer value (e.g., from '5' to 5)
- **`Integer.valueOf(str)`** converts a **String** value **str** into its corresponding integer value (e.g., "123" into 123)

Hint 3: Don't forget: the Luhn algorithm selects every other digit **starting at the final digit**, so be sure to test numbers with both even and odd lengths.

Hint 4: If you are having difficulty getting your program to work properly, try adding some **`System.out.println()`** statements to check the value of variables. **Don't forget to remove any such debugging statements before submitting!**

Hint 5: Be sure to include comments where appropriate in your code – particularly when implementing the Luhn algorithm check, things can get a bit complex, so be sure to leave whitespace between main sections of the code and to include descriptive comments.

Hint 6: It is fine to use additional "helper" methods if you choose to (although it is not required); just be sure to include all required methods in your submission file.

What to submit

Your final **checkCardNumber** method (and any helper methods if you used them) should be stored in a file **checkCardNumber.java.**

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JP2 moodle site. Click on Laboratory 2 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code – possibly **M:\Laboratory02** -- and drag only the following files into the drag-and-drop submission area:

- **computeScore.java**
- **checkCardNumber.java**

Then click the blue save changes button. Check the two .java files are uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your files and return feedback to you via moodle.

Outline Mark Scheme

Your tutor will mark your work and return you a score in the range "Excellent" (*****) to "Very poor" (*). We will automatically execute your submitted code and check its output; we will also look at the code to assess its general style before choosing a final mark.

Example scores might be:

5*: you complete both tasks correctly with no bugs, and your coding style is appropriate

4*: you complete one of the two tasks correctly and have made an attempt at the other, or you complete both tasks but the coding style is bad

3*: you completed only one of the tasks, or have made a reasonable attempt at both

2*: you have made some attempt at both tasks

1*: minimal effort